AFRL-IF-RS-TM-2003-3
In-House Technical Memorandum
February 2003

POLYPHASE FILTER AND DEMODULATION TECHNIQUES FOR OPTIMIZING SIGNAL COLLECTION PROCESSING

Alfredo Vega Irizarry

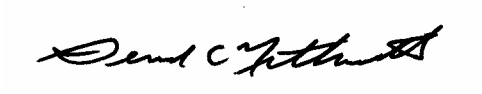
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

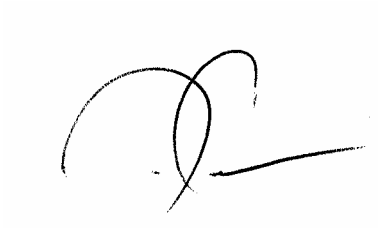
AFRL-IF-RS-TM-2003-3 has been reviewed and is approved for publication.

APPROVED:



GERALD C. NETHERCOTT
Chief, Multi-Sensor Exploitation Branch
Information and Intelligence Exploitation Division

FOR THE DIRECTOR:



JOSEPH CAMERA
Chief, Information & Intelligence Exploitation Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 2003	**In-House technical memo, Jun 02 – Jan 03**

4. TITLE AND SUBTITLE

POLYPHASE FILTER AND DEMODULATION TECHNIQUES FOR OPTIMIZING SIGNAL PROCESSING

5. FUNDING NUMBERS

PE - 62702F
PR - 459E
TA - PR
WU - OJ

6. AUTHOR(S)

Alfredo Vega Irizarry

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

AIR FORCE RESEARCH LABORATORY/IFEC
32 Brooks Road
Rome, NY 13441-4114

8. PERFORMING ORGANIZATION REPORT NUMBER

AFRL-IF-RS-TM-2003-3

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AIR FORCE RESEARCH LABORATORY/IFEC
32 Brooks Road
Rome, NY 13441-4114

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TM-2003-3

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Alfredo Vega Irizarry/IFEC/315-330-2382

12a. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

A polyphase filter for frequency demultiplexing as well as several demodulation techniques were explored and implemented to provide support to a signal collection system. An objective of the effort was to identify and implement those algorithms that exhibit optimum performance. A MATLAB and C++ implementation allowed testing and comparison of the polyphase filter method against a baseline. The implementation also allowed for comparison of synchronization using delays versus a phase-locked loop implementation. The project concluded with the creation of a graphical user interface that allows the user to display, create customized designs, and process the data.

14. SUBJECT TERMS war simulation, computer games

polyphase filter, demodulation, synchronization, PLL

15. NUMBER OF PAGES
44

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

List of Figures and Tables

List of Acronyms/Abbreviations

ADPLL	All Digital (Software) Phase Locked Loop
AM	Amplitude Modulation
BPF	Band Pass Filter
CBPF	Complex Bandpass Filter
DPLL	Digital Phase Locked Loop (Hardware Implementation)
DSB-SC	Double Sided Band Suppressed Carrier
DSP	Digital Signal Processing
DTLL	Digital Tanlock Loop
FDM	Frequency Division Multiplexed
FFT	Discrete Fast Fourier Transform
FFTW	Fastest Fast Fourier in the West
FIR	Finite Impulse Response Filter
FM	Frequency Modulation
GUI	Graphical User Interface
I	In-Phase
IIR	Infinite Impulse Response Filter
LPF	Low Pass Filter
NCO	Numerical Controlled Integrator
PLL	Phase Locked Loop (In General)
Q	Quadrature-Phase
SCP	Signal Collection Processing (Software)

1. Introduction

This document presents the development and implementation of a polyphase filter used for frequency demultiplexing as well as the evaluation of its performance compared to a baseline. A polyphase filter approach is an efficient method that performs frequency demultiplexing by means of decimation, filter reduction and the usage of the FFT algorithm. The development was made using Matlab and C++. The performance was measured in terms of execution speed as function of the filter parameters. The code has been embedded in a GUI that allows the user to go step by step in the design process, i.e., from filter design to audio extraction. The software also implements several demodulation techniques to complement the development.

2. Development

2.1 Conventional Method for Frequency Demultiplexing

The conventional method consists of a filtering and a downconversion stage in each frequency channel. (See Figure 2.1) In DSP, downconversion can be replaced by decimation which causes a similar effect as downconversion, but unlike downconversion, it does not require additional mathematical operations. Decimation has another benefit: it reduces the sampling rate and thus, reduces the size of the output data. The output data integrity will not be affected if the final sampling rate is equal or greater than the Nyquist rate of the filtered data.

2.2 Decimation

Decimation refers to a sampling rate reduction. For the purpose of this discussion, the decimation factor is an integer number N and decimation is done by retaining samples at time intervals of $N \cdot T_s$, $2N \cdot T_s$, $3N \cdot T_s$…

The frequency spectrum of a decimated signal can be express in terms of the original frequency spectrum.[1] The effect of decimation is the overlap (aliasing) of the original frequency spectrum at frequency intervals of $(F_s \div N)$, $2(F_s \div N)$, $3(F_s \div N)$… Since the power of the signal does not increase, the final spectrum has to be divided by a factor N. (See Equation 2.1)

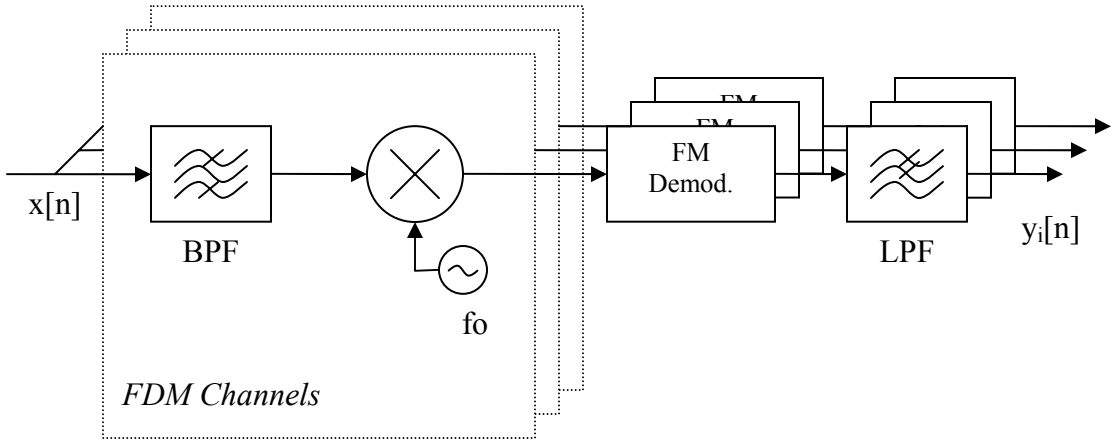


Figure 2.1 Signal Processing of FM Signals Using a Conventional Approach for FDM

$$S_d(e^{j\varpi}) = \frac{1}{N}\sum_{k=1}^{N} S(e^{j(\varpi - 2\pi\cdot k)/N}) \qquad (2.1)$$

Where S_d is the spectrum of the decimated signal S.

2.3 Polyphase Filter Method for Frequency Demultiplexing [2, 3]

A polyphase filter implementation reduces the computational inefficiencies of the conventional approach by means of decimating the input instead of the output, using a reduced filter bank and by applying the FFT algorithm.

A FIR filter impulse response h[n] is used for the development. FIR filters are preferred because they can be designed to preserve the phase and consequently would not cause distortion. The linear-phase FIR filter requires the impulse response to be symmetric or anti-symmetric around its middle point in the time axis.[4]

Consider the convolution between input x[n] and the filter impulse response h[n] producing an output $y[n] = x[n] * h[n]$. The concept of convolution can be explained using a convolution table or matrix as shown in Table 2.1.

Table 2.1 Convolution Table

n	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]
0	h[0]								
1	h[1]	h[0]							
2	h[2]	h[1]	h[0]						
3	h[3]	h[2]	h[1]	h[0]					
4	h[4]	h[3]	h[2]	h[1]	h[0]				
5	h[5]	h[4]	h[3]	h[2]	h[1]	h[0]			
6	h[6]	h[5]	h[4]	h[3]	h[2]	h[1]	h[0]		
7	h[7]	h[6]	h[5]	h[4]	h[3]	h[2]	h[1]	h[0]	
8	h[8]	h[7]	h[6]	h[5]	h[4]	h[3]	h[2]	h[1]	h[0]

The first row contains the sampled input signal at times 0, 1, 2…8. The first column represents the time sequence for which the output is calculated. The table is filled in with the filter coefficients.

The output $y[n]$ is given by the sum of products of filter coefficients in row n and the input $x[n]$; for example: $y[2] = h[2]\cdot x[0] + h[1]\cdot x[1] + h[0]\cdot x[2]$. Each column has the filter coefficient arranged in increasing order. (See the highlighted column, Table 2.1.)

Alternating samples are retained when the output is decimated by 2. The effect of decimation is shown in Table 2.2.

Table 2.2 Convolution Table, Decimation by 2

n	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]
1	h[1]	h[0]							
3	h[3]	h[2]	h[1]	h[0]					
5	h[5]	h[4]	h[3]	h[2]	h[1]	h[0]			
7	h[7]	h[6]	h[5]	h[4]	h[3]	h[2]	h[1]	h[0]	

Two filter patterns are formed after decimating the output. The patterns are made of arranging the even coefficient and the odd coefficients in ascending order. In addition, the filter made of the even coefficients only modifies the odd input sequence. The filter made of the odd coefficients modifies the even input sequence.

From a mathematical point of view, the decimated output can be modeled as a commutator, a decimator, two decimated filters made of even and odd coefficients and an adder, as shown in Figure 2.2. Equations (2.2a) through (2.2c) represent this process.

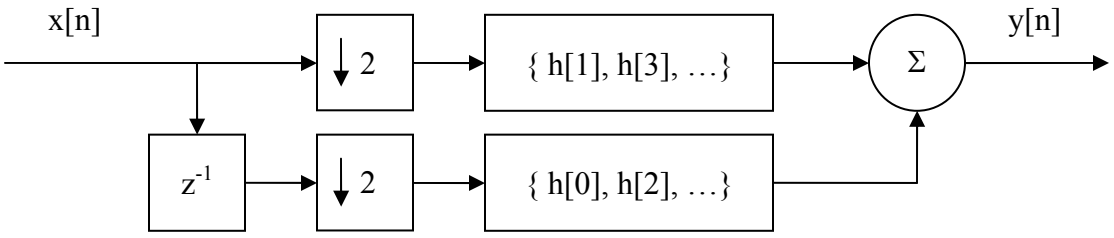


Figure 2.2 Equivalent Block Diagram When Decimation the Output by 2

$$y_0[2n] = x[2n+1] * h[2n] \tag{2.2a}$$

$$y_1[2n] = x[2n] * h[2n+1] \tag{2.2b}$$

$$y[2n] = y_0[2n] + y_1[2n] \tag{2.2c}$$

for n = 0,1,...

A similar analysis can be made when the output is decimated by *N*. (See Equation 2.3) The general block diagram representation is shown in Figure 2.3.

$$\begin{aligned} y[n] &= x[n] * h[n] \\ &= x[Nn + N - 1] * h[Nn] \\ &\quad + x[Nn + N - 2] * h[Nn + 1] \\ &\quad \ldots \\ &\quad + x[Nn] * h[Nn + N - 1] \end{aligned} \tag{2.3}$$

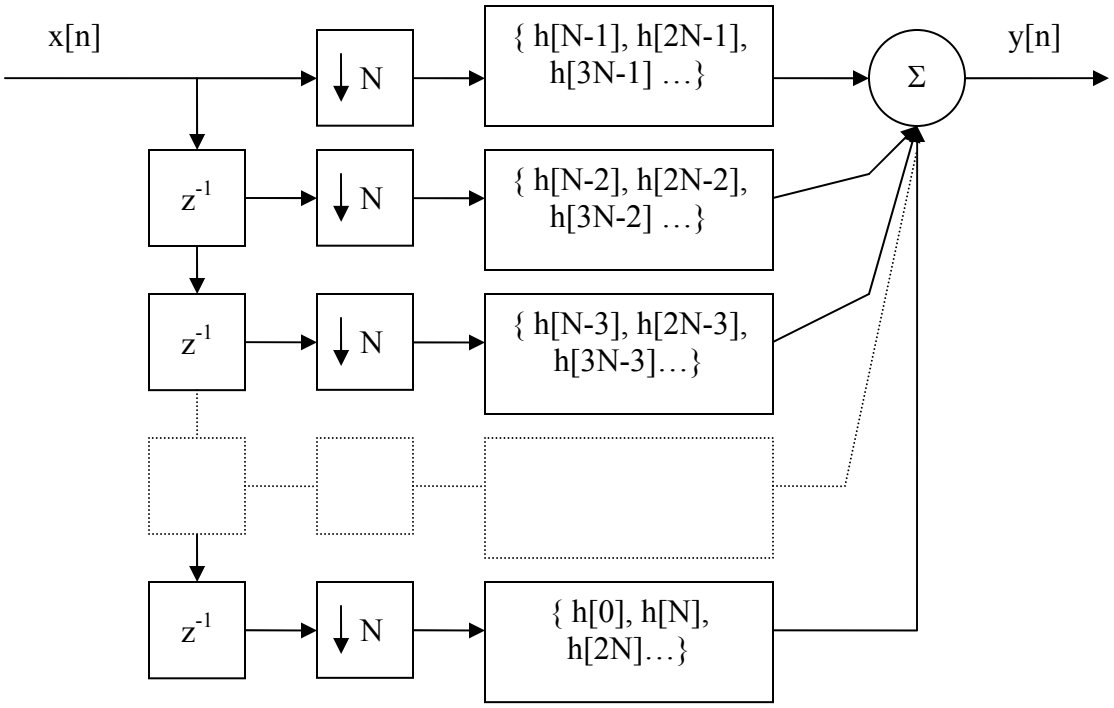


Figure 2.3 Equivalent Block Diagram When Decimating the Output by N

The polyphase filter architecture of Figure 2.3 has some of the computational advantages:

- First, decimation has reduced some computational effort by reducing the number of input samples.
- Second, decimation causes aliasing of the output signal. The frequency spectrum of the signal is periodic and it reflects to baseband. A frequency rotation might be needed to center the spectrum on the frequency axis. However, this process can be ignored in the present development because the demodulation schemes used after frequency demultiplexing do not require the rotation.

The polyphase filter method requires the design of a LPF model filter. Let h_0 represents the input response of a low pass filter. The variable L represents the number of filter taps and it is a multiple of the number of channels N.

$$\{ h_0[0] , h_0[1], h_0[2], h_0[3], \ldots, h_0[L\text{-}1] \}$$

For frequency demultiplexing, filters $h_i[n]$ are used for channel i. The filter responses are of equal magnitude. Each filter is a rotation of the original filter, uniformly shifted in frequency. (See Figure 2.4) The resulting filters $h_i[n]$ form a bank of complex bandpass filters.

$$h_i[n] = h_0[n] \cdot e^{j 2\pi \cdot i \cdot n / N} \tag{2.4a}$$

$$H(e^{j\omega}) = H_0(e^{-j\varpi - 2\pi \cdot n / N}) \tag{2.4b}$$

Where:

N = number of channels,

i = channel number,

n = sequence number.

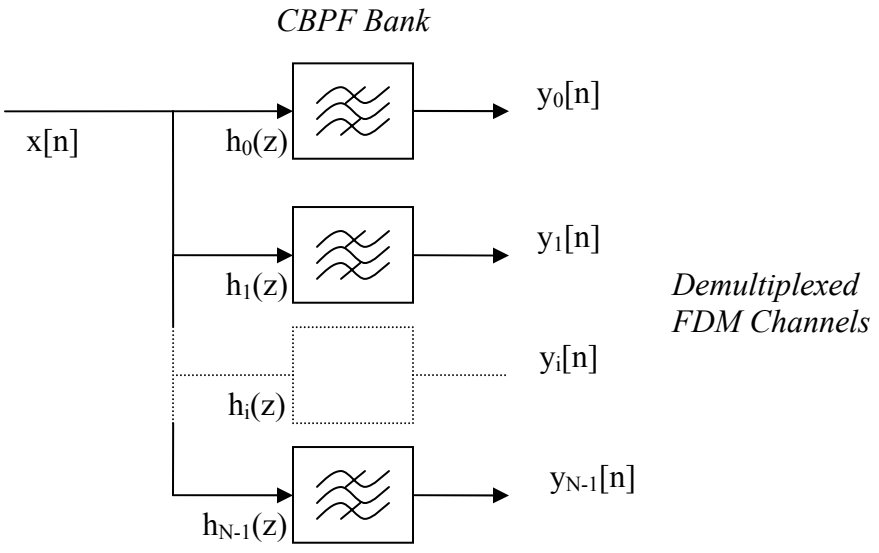


Figure 2.4 FDM Demultiplexer Block Diagram

The polyphase filter development requires that the output be decimated by a factor equal to the number of channels, *N*. (See Equation 2.5) A further development tries to exploit the symmetries of weight function *W*. [5] (See Equation 2.6)

$$y_i[n] = x[n] * h_i[n] \tag{2.5}$$

for n=0, 1, 2, … N

$$W^{in} = e^{-j2\pi \cdot n \cdot i / N} \tag{2.6}$$

The convolution table for channel *i* is shown Table 2.3.

Table 2.3 Effects of Decimation on a Uniform Filter Bank

n	x[0]	x[1]	x[2]	x[3]
N-1	$h[1N-1]\cdot W^{-i(1N-1)}$	$h[1N-2]\cdot W^{-i(1N-2)}$	$h[1N-3]\cdot W^{-i(1N-3)}$	$h[N-4]\cdot W^{-i(1N-4)}$
2N-1	$h[2N-1]\cdot W^{-i(2N-1)}$	$h[2N-2]\cdot W^{-i(2N-2)}$	$h[2N-3]\cdot W^{-i(2N-3)}$	$h[2N-4]\cdot W^{-i(2N-4)}$
3N-1	$h[3N-1]\cdot W^{-i(3N-1)}$	$h[3N-2]\cdot W^{-i(3N-2)}$	$h[3N-3]\cdot W^{-i(3N-3)}$	$h[3N-4]\cdot W^{-i(3N-4)}$
4N-1	$h[4N-1]\cdot W^{-i(4N-1)}$	$h[4N-2]\cdot W^{-i(4N-2)}$	$h[4N-3]\cdot W^{-i(4N-3)}$	$h[4N-4]\cdot W^{-i(4N-4)}$
5N-1	$h[5N-1]\cdot W^{-i(5N-1)}$	$h[5N-2]\cdot W^{-i(5N-2)}$	$h[5N-3]\cdot W^{-i(5N-3)}$	$h[5N-4]\cdot W^{-i(5N-4)}$
6N-1	$h[6N-1]\cdot W^{-i(6N-1)}$	$h[6N-2]\cdot W^{-i(6N-2)}$	$h[6N-3]\cdot W^{-i(6N-3)}$	$h[6N-4]\cdot W^{-i(6N-4)}$

Because the weight function is periodic in nature, it can be demonstrated that the following property is valid.

$$W^{-i(Nn-m)} = W^{-i(N-m)} \tag{2.7}$$

Table 2.4 shows the results of substituting Equation 2.7 into Table 2.3. This step allows the simplification of the filter bank of Figure 2.4 into what is known as the Polyphase Filter Method for Frequency Demultiplexing. Each column contains the decimated filter coefficients multiplied by a constant gain or weight given a particular FDM channel. Thus, the equivalent block diagram (See Figure 2.5) has *N* sub-filters followed by *N* gain states.

Table 2.4 Effect of Decimation on a Uniform Filter Bank

n	x[0]	x[1]	x[2]	x[3]
N-1	$h[1N-1]\cdot W^{-i(N-1)}$	$h[1N-2]\cdot W^{-i(N-2)}$	$h[1N-3]\cdot W^{-i(N-3)}$	$h[1N-4]\cdot W^{-i(N-4)}$
2N-1	$h[2N-1]\cdot W^{-i(N-1)}$	$h[2N-2]\cdot W^{-i(N-2)}$	$h[2N-3]\cdot W^{-i(N-3)}$	$h[2N-4]\cdot W^{-i(N-4)}$
3N-1	$h[3N-1]\cdot W^{-i(N-1)}$	$h[3N-2]\cdot W^{-i(N-2)}$	$h[3N-3]\cdot W^{-i(N-3)}$	$h[3N-4]\cdot W^{-i(N-4)}$
4N-1	$h[4N-1]\cdot W^{-i(N-1)}$	$h[4N-2]\cdot W^{-i(N-2)}$	$h[4N-3]\cdot W^{-i(N-3)}$	$h[4N-4]\cdot W^{-i(N-4)}$
5N-1	$h[5N-1]\cdot W^{-i(N-1)}$	$h[5N-2]\cdot W^{-i(N-2)}$	$h[5N-3]\cdot W^{-i(N-3)}$	$h[5N-4]\cdot W^{-i(N-4)}$
6N-1	$h[6N-1]\cdot W^{-i(N-1)}$	$h[6N-2]\cdot W^{-i(N-2)}$	$h[6N-3]\cdot W^{-i(N-3)}$	$h[6N-4]\cdot W^{-i(N-4)}$

The computational advantages of the polyphase filter also include:

- Computational effort is saved since each channel uses similar filter-bank calculations. First, convolution is done for each decimated input and decimated filter. These steps are the same for all the channels.
- When the filter outputs are combined to recover the channels, the weight functions form the Inverse Discrete Fourier Transform Matrix (See Equation 2.8). This fact suggests that speed performance can be achieved if the number of channels, *i*, is a power of two. In this case, the Discrete Fourier Transform can be replaced with the Inverse Fast Fourier Transform.

$$y_i = \sum_{l=0}^{N-1} W^{-il} v^l \qquad (2.8)$$

For i = 0, 1… N-1

The variable v_l is the output of the polyphase filter *l.*

2.5 Polyphase Filter Demultiplexing Algorithm

Pre-filtering Steps

1. Design a FIR low pass filter *h(n)*. The number of taps *L* should be a multiple of the number of channels *N*.
2. Decimate *h* by a factor equal to the number of channels. The result is a set of decimated filters h_i, where $i = 0,1,\ldots N-1$ channels.

Commutator Operation

3. Decimate the input x by a factor equal to the number of channels. The result is an N dimensional vector of decimated data $x_{i,}$ where $i = 0, 1 \ldots N\text{-}1$.

Filtering Operations

4. Filter the decimated input using the decimated filters. Notice that the decimated filter h_i has to operate on the decimated input x_{N-i+1}. The output of the filters will be a set of vectors called v_i.

Fourier Transform operation

5. Arrange the vectors v_i in a matrix m_{ij}, where each matrix row i represents column vector v_i. (See Equation 2.8)
6. Compute the Inverse of the Fourier Transform of each column j of the matrix m_{ij}. The Inverse of the Fourier transform is the matrix $[W_{ij}]$ where W_{ij} is the weight function in Equation 2.6. The product of the two matrices is N row vectors that represent the output of each channel.

2.6 Efficiency Measurements

The performance of the conventional and polyphase filter implementation will be measured in terms of their processing time.

It has been claimed that the *conventional approach* is computationally inefficient compared to the polyphase filter method. First, the outputs from each channel are oversampled relative to the channel bandwidth when downconversion is used. This can be corrected using decimation as it has been explained before. However, the approach still requires the multiple numbers of calculations that comes from convolving the input signal with each channel filter.

Whether using decimation or not, the overall number of floating point products/divisions in a non-optimized real time execution is expressed by Equation 2.9. [2]

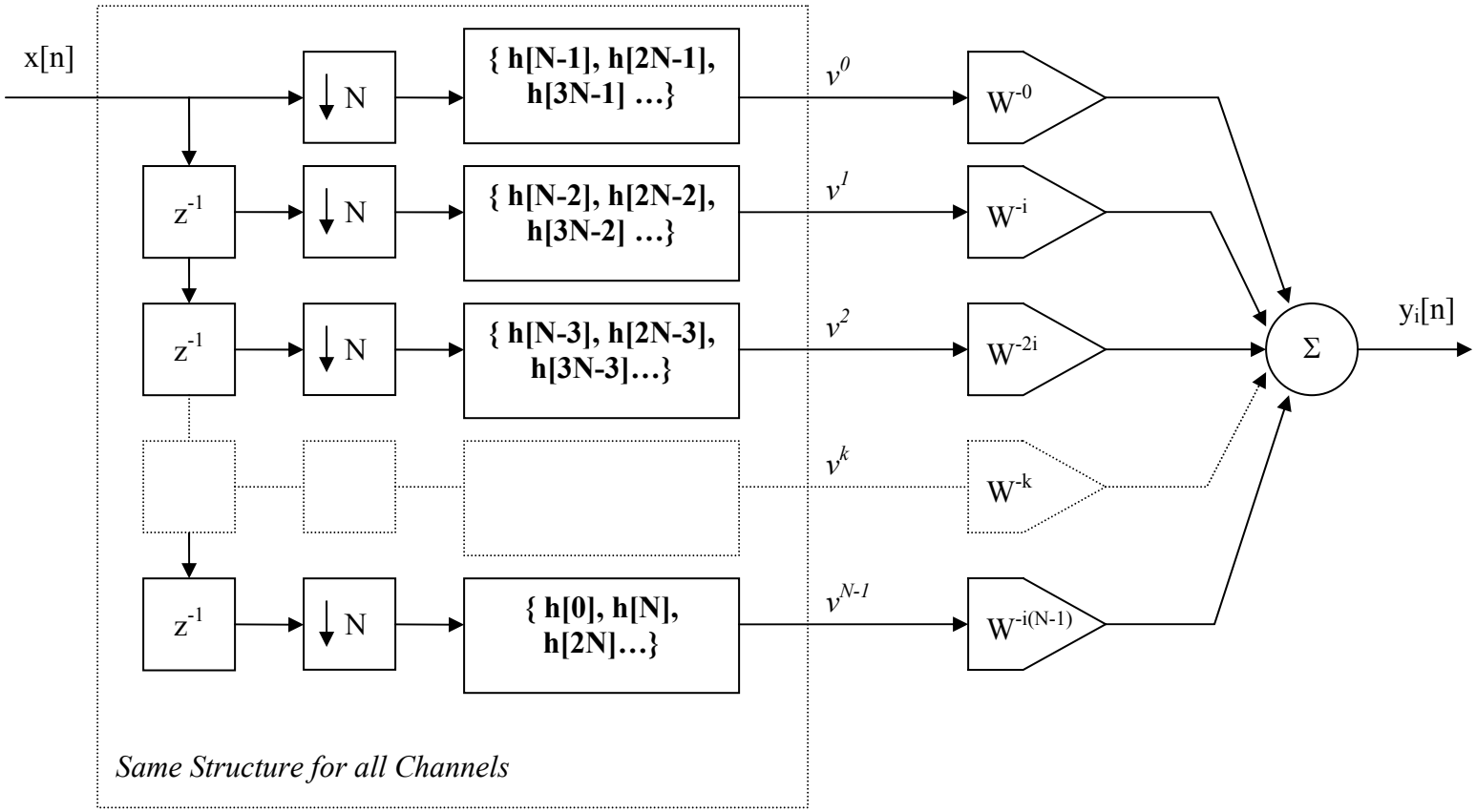


Figure 2.5 Polyphase Filter Block Diagram for Channel i

$$Operations/time = N \cdot L \cdot f_s \quad (2.9)$$

Where:

N is the number of channels;

L, the number of taps and

f_s, the sampling rate.

In the case of the *polyphase filter*, the number of operations per time is given by Equation 2.10. [2]

$$Operations/time = [L + \log_2(N)]f_s \quad (2.10)$$

Where:

N is the number of channels,

L the number of taps and

f_s is the sampling rate.

The logarithmic term in Equation 2.10 is due to the convergence rate of the FFT algorithm. Note that in the case of the polyphase filter, when the number of channels is equal to 1, Equations 2.9 and 2.10 are identical. So it can be stated that the conventional method is a particular case of the polyphase filter method.

The polyphase filter can be seen as an optimization of the conventional approach. Moreover, the polyphase filter can be optimized even further. Its optimization relies on the optimization of the FFT and filtering methods that will be discussed in the evaluation of the conducted tests.

The above equations assume that the conventional algorithms are used for filtering and FFT. The polyphase filter method can be much faster depending on their respective degree of optimization.

3. Polyphase Filter Implementation

The purpose of developing the polyphase filter is to support a particular signal collection system as illustrated in Figure 3.1. The configuration consists of an antenna

(or antenna array); a (WJ) receiver with a programmable SCSI connection, 8 channels for analog and digital outputs; and a PC with an internal ADC.

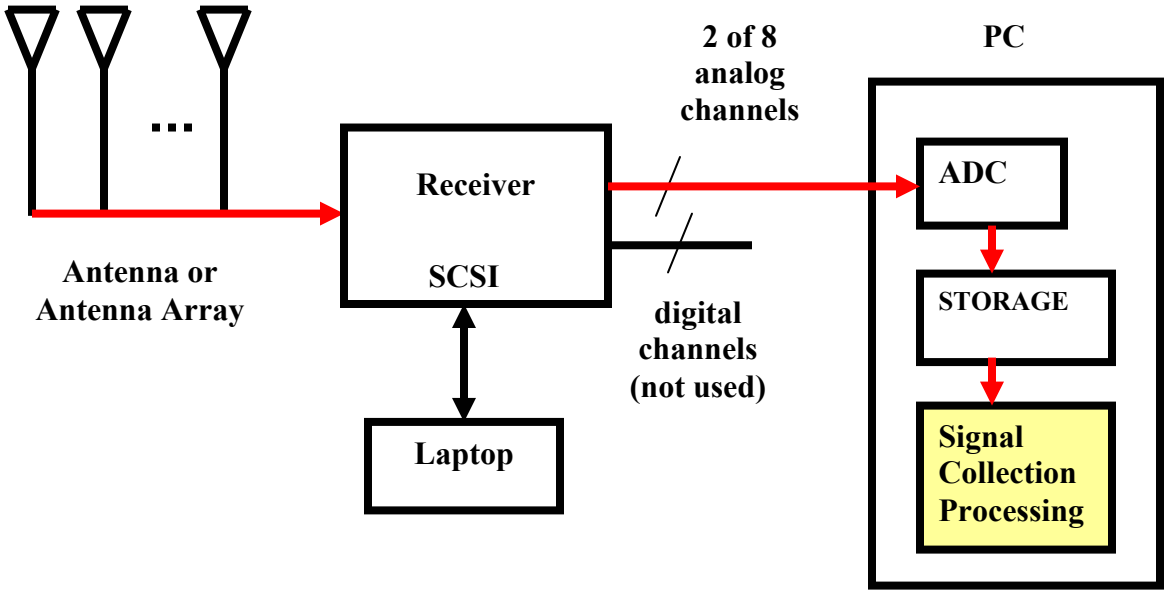


Figure 3.1 Configuration for Data Collection

In this application, the antennas receive signals from different sources of opportunity. The receiver downconverts to an IF and provides filtering for those signals that are of interest in our chosen application: commercial FM radio stations. A computer receives samples of the analog output at a rate of up to $f_s = 6.4$ MHz and stores the samples in memory. An ICS 650 dual channel ADC is used. The size of each file in memory is limited and depends on the selected sampling rate and storage type (memory or disk). Once collected, files on order of 650 MB can be stored to a CDR optical disk.

The polyphase filter will be part of a collection of subroutines used to process signals of interest. The final goals of this effort are to:

1. Achieve Frequency Demultiplexing
2. Achieve Computational Efficiency
3. Develop a portable code, and
4. Embed the code in a user-friendly GUI.

4. Testing and Debugging

Testing will ensure proper performance of the polyphase filter algorithm. Testing is based on:

- **Extracting the audio signal.** This is the ultimate goal and it indicates that the whole algorithm is working properly. This test requires the implementation of a demodulation process that will be discussed later on;
- **Measuring speed relative to the conventional approach.**

Other tests performed at a low level of development are:

- **Testing the polyphase filter using single harmonic signals.** This test will be able to determine if the polyphase filter is demultiplexing the signal as expected;
- **Testing the polyphase filter using an impulse signal.** By applying an impulse response, each channel should reproduce the filter coefficients.

After successful audio extraction, the next step required is to compare performance of the polyphase method against the conventional approach. Because the polyphase filter is implemented on a system that is not dedicated to the algorithm, deviations from Equations 2.9 and 2.10 occur and should be expected. The results comparing the C++ and Matlab implementations are shown in the following figures.

Figure 3.2 shows measured data of a polyphase filter algorithm implemented in Matlab and C++. The results show a linear relationship between processing time and the number of filter taps. The data was measured in a 2.1GHz, 512MB RAM, Dual Processor, Dell Computer™, with Windows XP OS.

The linear relationship between processing time and filter tabs is predicted by Equation 2.9 and 2.10. It can be seen that Matlab is using some kind of optimization. It was observed that a 32-point Matlab FFT algorithm runs approximately 10 times faster than the standard 32-point FFT in C++. This fact can explain the reason for having an approximate 10:1 ratio between the slope of the blue and red traces of Figure 4.1.

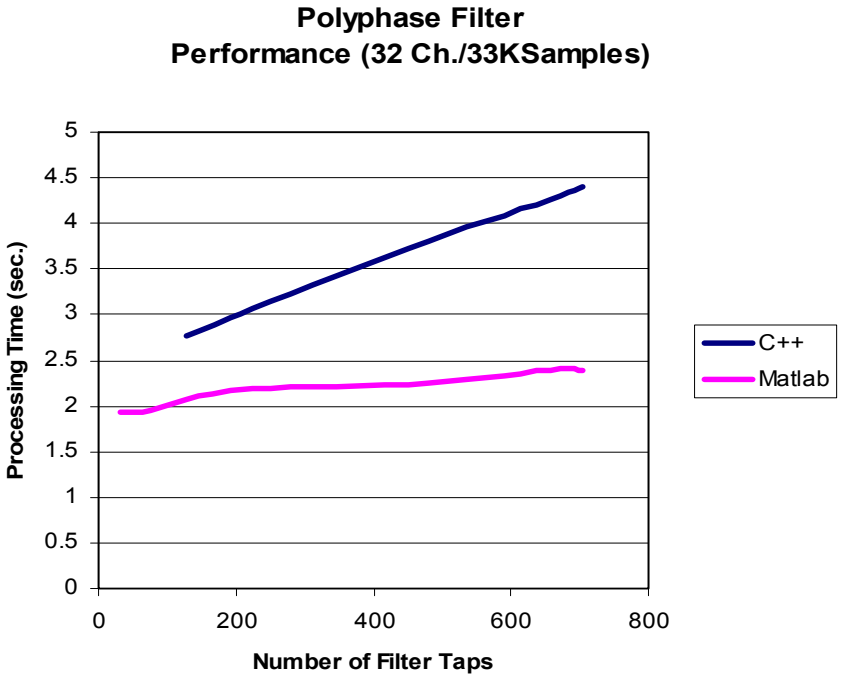


Figure 4.1 Processing Time vs. Filter Taps

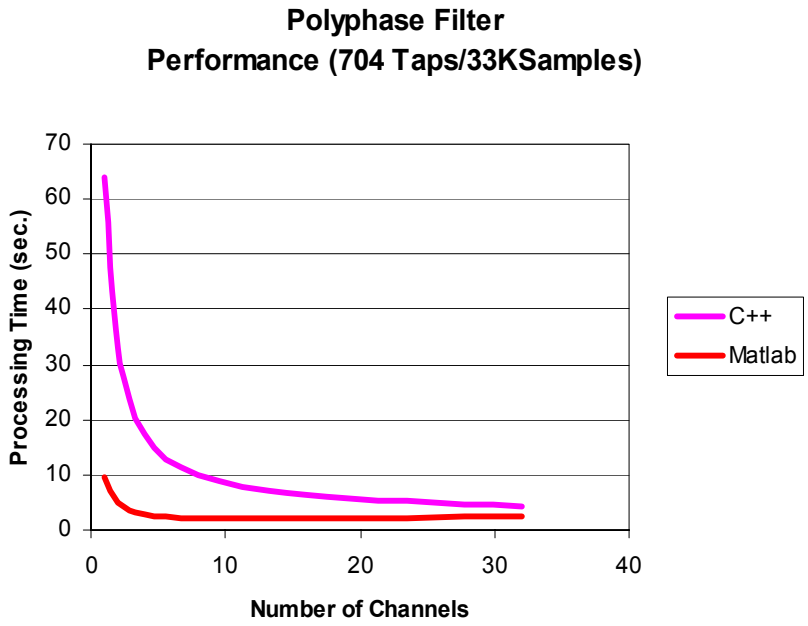


Figure 4.2 Processing Time vs. Number of Channels

Figure 4.2 suggests that the processing time is faster than the predictions and its performance seems to be better than expected. Under the assumptions of Equations 2.9 and 2.10, the processing time should have a logarithmic relationship. The causes for this behavior are not well known and several propositions will be made in Section 5.

Additional testing was performed to compare the Matlab and C++ subroutines. Since Matlab seems to have a superior performance than C++, the particular process speed was measured. (See Figures 4.3-4.5)

The most considerable difference is found in the FFT process that Matlab uses. In the polyphase filter algorithm the number of point of the FFT is equal to the ratio of number of filter taps and the number of channels. Having a small number of channels implies calculating a large number of FFT points. The speed of Matlab FFT algorithm is approximately 10 times faster than the standard FFT.

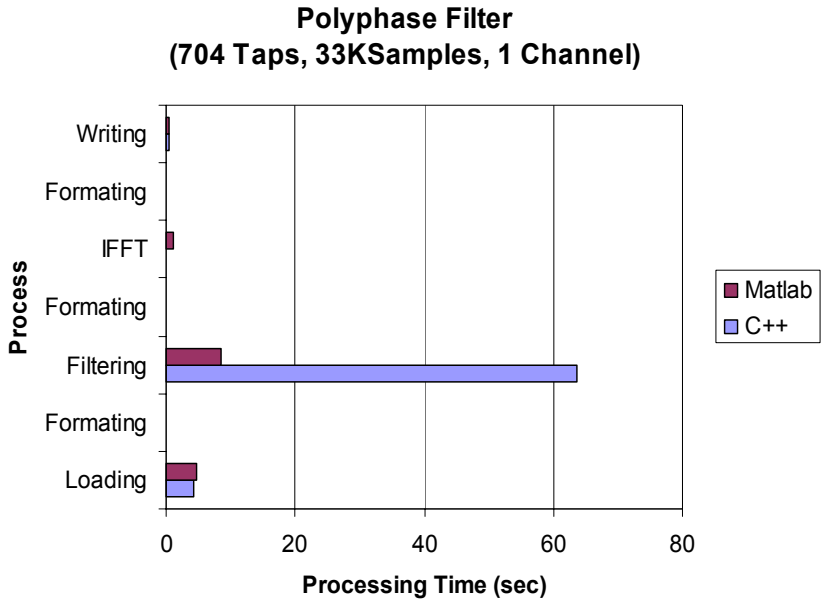


Figure 4.3 Processing Time for One Channel

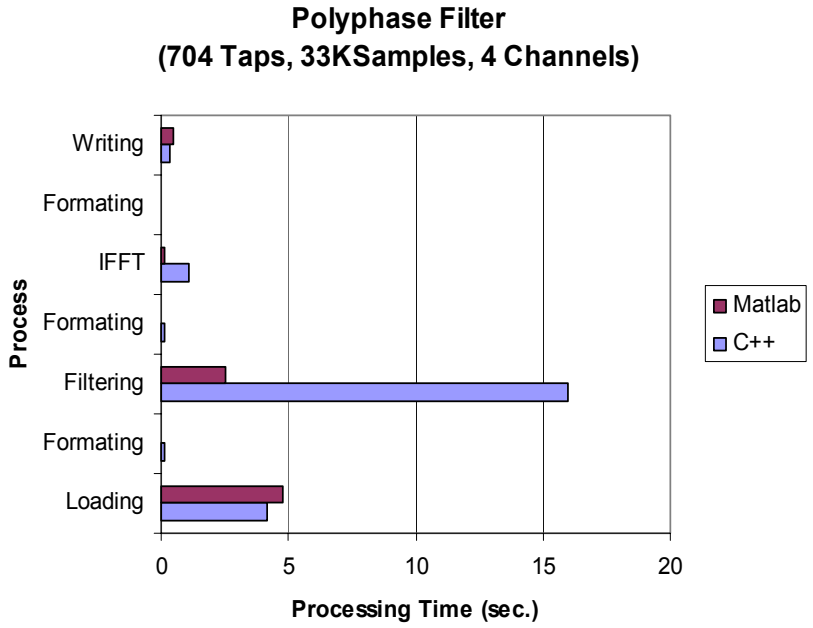


Figure 4.4 Processing Time for Four Channels

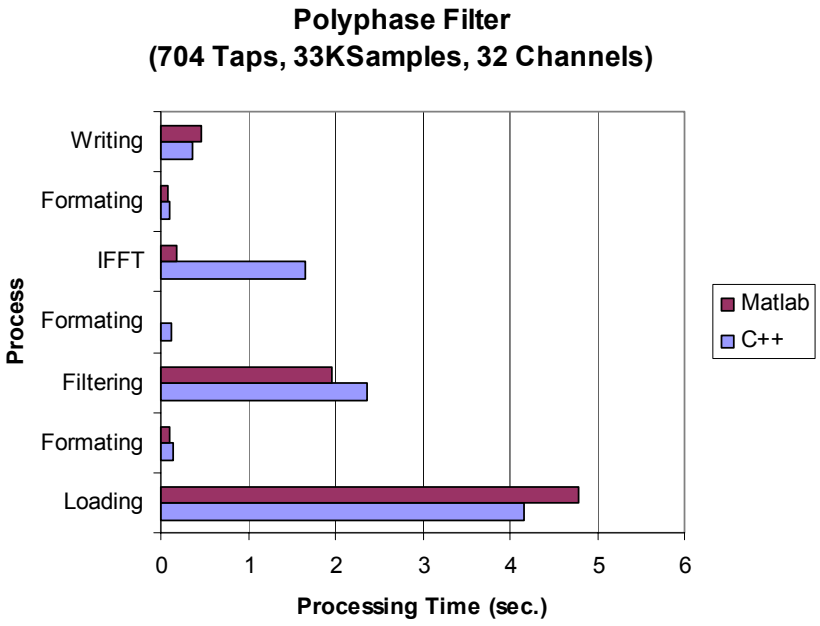


Figure 4.5 Processing Time for 32 Channels

The filtering algorithm is responsible of creating another significant difference between Matlab and the C++ code. The worst case scenario occurs for large number of filter taps. It is also noticed that this process that takes the most amount of the time, excluding reading and writing operations.

5. Evaluation of the Polyphase Filter Method

A graph showing the processing time per channel versus the number of channels confirms the efficiency of the polyphase filter method compared to the conventional approach. (See Figure 5.1) The graph shows a baseline, i.e. the conventional approach, based on a Matlab algorithm. For a fixed tap filter, increasing the channel number does not change the processing time per channel. Conversely, changing the number of channels in the polyphase filter approach will reduce the processing time per channel.

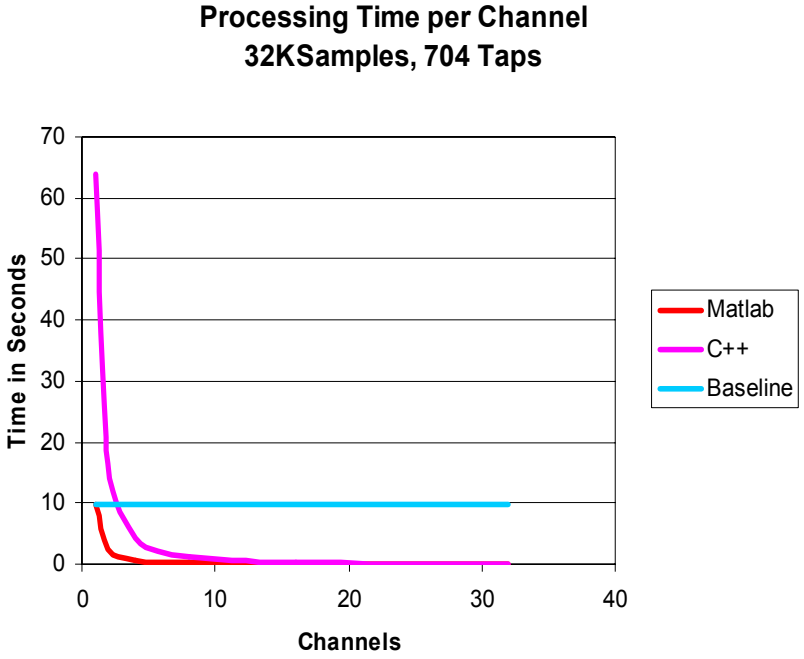


Figure 5.1 Estimated Processing Time per Channel

The polyphase filter algorithm becomes exceedingly fast relative to the conventional method when the number of channels is large. Since the polyphase

algorithm relies on the FFT and filtering routines, it is expected that its overall performance depends on the degree of optimization of the FFT and filtering algorithms.

After investigating the possible causes that makes Matlab to exhibit better performance than the C++ algorithm, it can be stated that:

- Matlab uses a *FFTW* algorithm which proved to be faster than the traditional FFT algorithm. The *FFTW* package was developed at Massachusetts Institute of Technology by Matteo Frigo and Steven G. Johnson. [6]
- Matlab can be using FFT-convolution methods, also called *overlap-add method* and *overlap-save method,* for their filtering. These methods are based on multiplying the frequency spectrum of the impulse response and the signal and then transforming into the time domain. They also require zero padding or data truncation respectively. An approximate number of products/divisions required in this algorithm is given by Equation 5.1 [7]

$$\frac{L^2}{8 \cdot (3 \cdot L \cdot \log_2 L + L)} \tag{5.1}$$

Where L is the number of taps.

6. Related Efforts

Demodulation schemes were developed as part of the polyphase filter development and implementation. They were needed to recover the audio of the demultiplexed signals. The following sections discuss the implementation of those methods.

6.1 FM Demodulation

The quadrature components of the FM signal were used to recover the audio signal. These types of signals produce simple results for FM and AM demodulation. The quadrature components I and Q are given by Equations 6.1a-c. The complex signal formed by I and Q has the peculiarity of having only non-negative spectral components.[8]

$$s[n] = I[n] + j\, Q[n] \tag{6.1a}$$

$$I = f(nT) \cos(2\pi f_0 nT) \tag{6.1b}$$

$$Q = f(nT)\sin(2\pi f_0 nT) \quad (6.1c)$$

The procedure for digital signals requires the following steps: [9]

- Obtaining the rate of change of the signal in the time domain.
    - This is obtained by multiplying the conjugate of the delayed signal by the signal itself. The product $x[n]$ contains the audio information in the phase which is proportional to the frequency change.

$$x[n] = y[n] \times y[n-1]^* \quad (6.2)$$

- Calculating the phase difference of the signal.
    - Signed arctan function is used to extract the phase of x

$$s[n] = \text{Arctan}(\, x[n]\,) \quad (6.3)$$

- Filtering.
    - For commercial FM radio stations the filter bandwidth is 19 kHz.

6.2 FM Stereo Demodulation

The modulation of an FM stereo signal requires:

FM Demodulation

- Obtaining the rate of change of the signal in the time domain.
- Calculating the phase difference of the signal as in Equation 6.3.

AM-DSB-SC Demodulation

The final signal $z[n]$ is composed of three spectral signals. The frequency spectrum will show:

L+R (left plus right) component 0-15 kHz*

DSB-SC component	23-53 kHz*
Pilot tone	19 kHz*

*For FM stereo radio stations [8, 10]

The general idea is to demodulate the DSB-SC component using the pilot tone. The demodulated signal is the L-R component that is needed to recover the left (L) and right (R) channels of a stereo signal. In this process, the filtered L-R component can lose phase reference to the L+R component. Adding and subtracting the L-R and L+R components will not result in the recovery of the channels unless their phases have been synchronized.

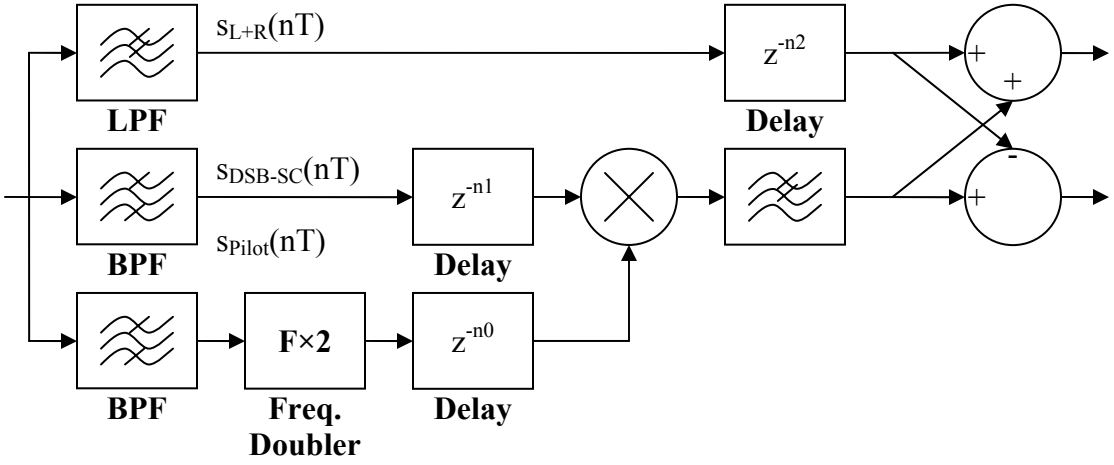


Figure 6.1 FM Stereo: DSB-SC Demodulation and Synchronization

6.3 Synchronization [9]

As mentioned before, FM demodulation produces a signal with three spectral components: a pilot tone, L+R and DSB-SC. (See Equation 6.4)

$$s[nT] = s_{Pilot}[nT] + s_{L+R}[nT] + s_{DSB-SC}[nT] \qquad (6.4)$$

For FM Stereo demodulation, the envelope of the term $s_{DSB-SC}[nT]$ needs to be recovered without loosing phase reference with respect to $s_{L+R}[nT]$. A filter is required to

extract the component. However, these filters add a delay to each signal causing the components to lose phase reference.

Output LPF: $s_{L+R}[n-n_{LPF}]$, delay = n_{LPF} (6.5a)

Output BPF: $s_{DSB-SC}[n-n_{BPF}]$, delay = n_{BPF} (6.5b)

Output BPF: $s_{Pilot}[n-n_{NBPF}]$, delay = n_{NBPF} (6.5c)

Additional delays (See Equations 6.5a-c) have to be added in order to maintain phase reference relative to each other. (See Equations 6.6a-c)

Output L+R: $s_{L+R}[n-n_3] = s_{L+R}[n-n_{LPF}-n_2]$ (6.6a)

Output L-R: $s_{L-R}[n-n_3] = s_{L-R}[n-n_{BPF}-n_1]$ (6.6b)

$s_{L-R}[n-n_{BPF}- n_1] = s_{Pilot}[n-n_{NBPF}- n_0]$ (6.6c)

Where: n_3 is the total delay

6.4 Synchronization using PLL

A PLL design simplifies the process of synchronizing the FM constituents, but requires a considerable effort to understand its design and operation.

A motivation for using a PLL to recover the channel L and R is that this device can lock to the pilot tone without requiring previous filtering. It has been shown that the PLL has excellent noise performance and thus, it is expected not to be susceptible to the spectral components L+R and AM-DSB-SC. The output of the PLL, i.e., the locked signal, is doubled in frequency to demodulate the AM-DSB-SC component. The overall process shown in Figure 6.2 does not require delay blocks to maintain phase reference. As a trade off, the PLL requires a short time to lock to the signal. This time is known as settling time and constitutes an important design parameter. [11, 12]

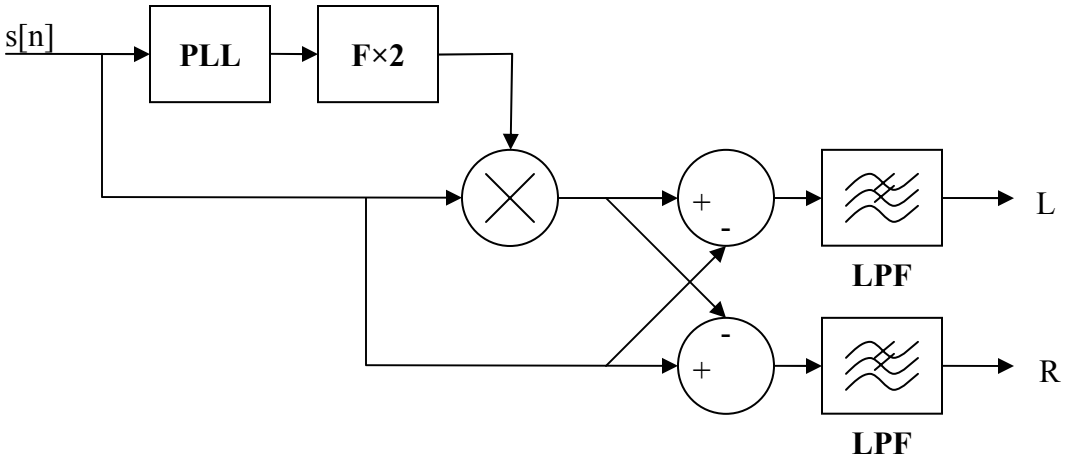


Figure 6.2 Channel Recovery from a Stereo Signal Using a PLL

6.4.1 ADPLL Design

PLL literature distinguishes between analog PLL, digital PLL (DPLL) and software PLL (ADPLL). For the purpose of this discussion, PLL can designate any of these types. The DPLL usually refers to a hardware implementation of a digital design, so it will be irrelevant to the present discussion. [11]

Several methodologies were explored for the PLL design. Table 6.1 compares the design considerations of each approach.

Table 6.1 Design Considerations of PLLs

PLL Type		Analog PLL	ADPLL	Costas ADPLL	DTLL
Non-linearities	$\sin(\theta)$	✓	✓	✓	
	Loop Gain as function of amplitude of input signal	✓	✓	✓	
Stability Issues	Poles and Zeros of LPF	✓	✓		
Assumptions	≈90° phase difference between the input and the locked signal	✓	✓	✓	✓

PLL Type		Analog PLL	ADPLL	Costas ADPLL	DTLL
Tracking the Input	Impulse, step, ramp responses	✓			

The first approach consisted of designing an analog PLL and then converting it to an ADPLL. This approach proved to be extremely difficult and inappropriate because the transformation to the discrete domain seldom generates stable designs. Even if the poles and zeros of the PLL's characteristic equation fall inside the unit circle (a stability condition), the root locus could leave the circle for particular loop gain values as shown in Figure 6.3.

Another explored method consisted of designing the PLL in the discrete domain. The method did not differ much to the previous one. It was found that using a high order LPF in the ADPLL can contribute to instability for high loop gain values. In the PLL implementation of Figure 6.3, the loop gain depends on the amplitude of the input signal. There is the possibility of having unstable responses usually for high amplitudes and especially when the PLL transfer function contains multiple poles and zeros. Several examples found in literature develop ADPLL using transfer functions with few poles and zeros, presumably because adding extra poles and zeros causes more stability problems. [13]

A third method is based on the Costas PLL shown in Figure 6.4. Two multipliers are used to combine the I and Q components of the NCO with the input signal. The design has been proven to work, although it is susceptible to nonlinearities of the sine function. In this approach the filtering is done by canceling the unwanted terms rather than using a LPF.

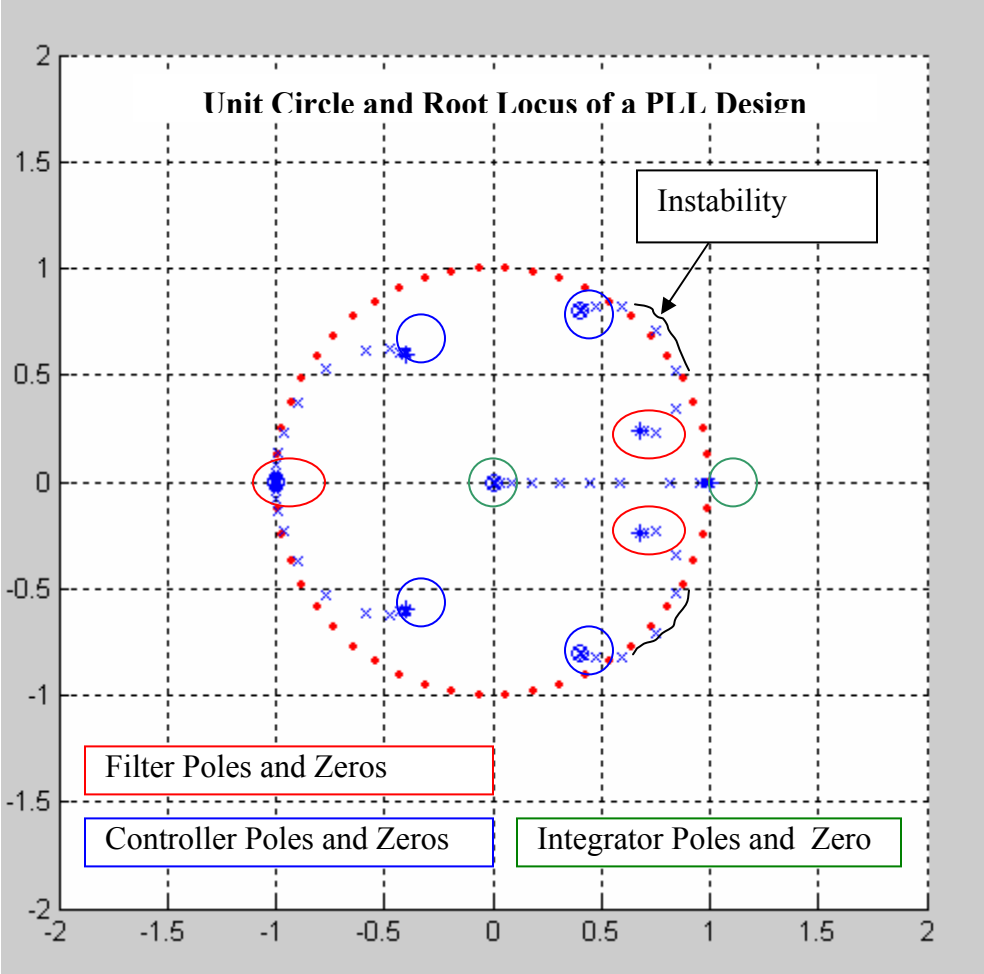


Figure 6.3 Root Locus of a PLL Design

A fourth approach attempts to eliminate all the non-linearity of the Costas Loop. The design is known as a digital tanlock loop (DTLL). The design adds the arctan(y,x) function to extract the phase difference. In this design, the phase detector (a non-linear component) and the arctan(y,x) function combined together can be represented as a linear component. Some adjustments were made to extend the output range of the arctan function beyond $\pm\pi$. It also eliminates the effects of the amplitude of the input signal in the loop. Another advantage of this approach is that adding a high order LPF in between the multiplier and the arctan function will not contribute to the poles and zeros of the ADPLL characteristic equation. An illustration of this idea is shown in Figure 6.5.

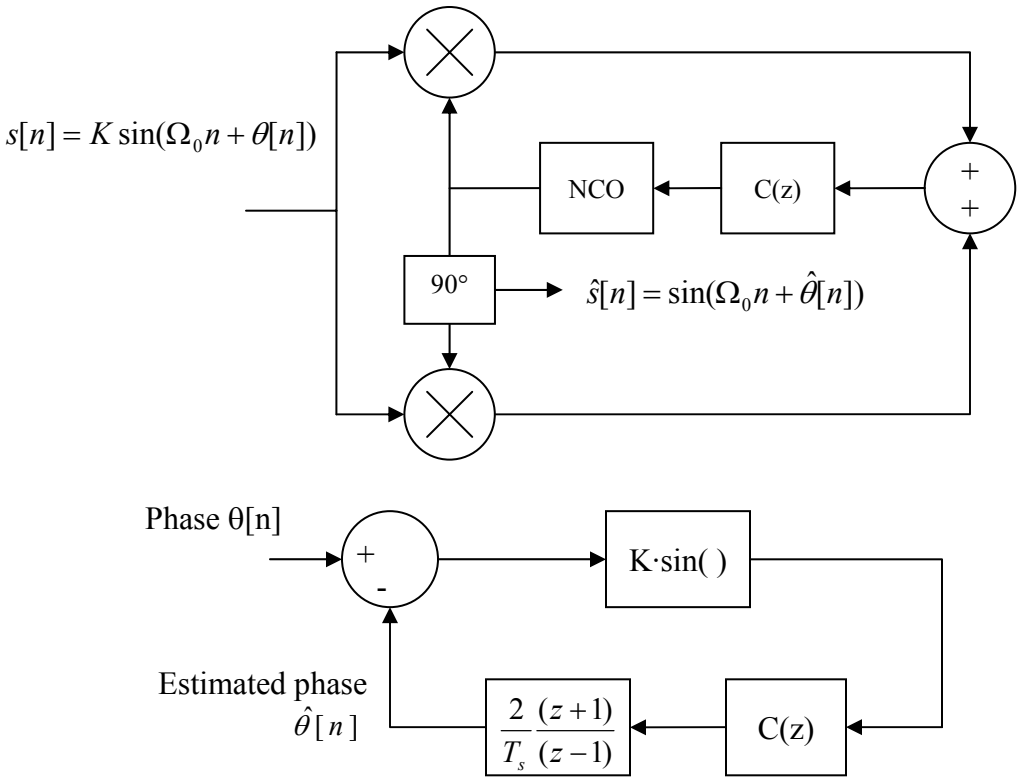


Figure 6.4 Costas Loop and Equivalent Control System Diagram

A disadvantage of using an ADPLL against the filter approach of Section 6.2 is that the algorithm requires each block output to be calculated separately at a specific time due to the nonlinear calculations and the presence of blocks with memory. The Matlab algorithm was found to be 10 times slower than baseline approach.

A discussion of the stabilization techniques can be found in references [11-14]. The most common approach is to start the design based on a second order analog system and translate it to the discrete domain. [12]

Tracking considerations were found to be irrelevant to the ADPLL design. A one order PLL was able to track different input signals. Finding an explanation was out of the scope of this study, but in theory a second order ADPLL should be able to follow exponential signals due to the term (z-1) in the error transfer function. [13]

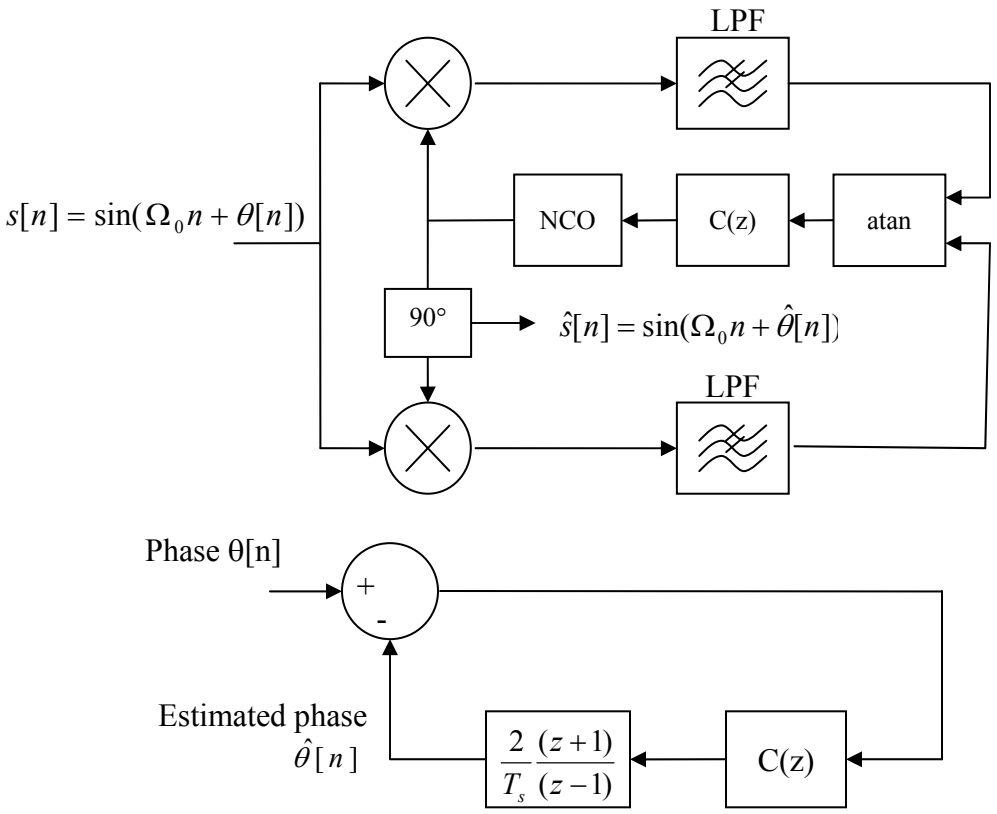


Figure 6.5 Digital Tanlock Loop and Equivalent Control System Diagram

6.5 AM Demodulation

Demodulating a discrete AM signal is relatively a simple process. (See Equation 6.8) When the in-phase and quadrature components are provided, the AM demodulation consists of calculating the amplitude of the complex signal. [9, 15]

$$y[n] = \text{abs}(\, x[n]\,) \qquad (6.8)$$

7. SCP Software

SCP or Signal Collection Processing Software is the final product of the efforts presented in this report. SCP is a program written in Matlab 6.5 and C++ 6.0 that allows the user to process communication signals using the demultiplexing and demodulation techniques presented in this report.

7.1 Software Architecture

Matlab provides limited software capabilities for GUI design. Some of the functionality of the Signal Collection Processing Software, *SCP*, comes from Windows® programming. The software makes use of two ActiveX components: *TreeView* and *Grid* Controls.

The *SCP* has three main components: a *Tree View* control, a *Display* and *Navigation Buttons*.

The *TreeView* control displays the menu. The menu items are ordered following the design process, from *data loading*, *data display*, *filter design*, *data processing*, and *audio extraction*. The *TreeView* also allows for displaying submenus. For instance, the *filter design* menu has two submenus: one called *channel finder* and the other *channel adjust*. They allow the user to estimate the number of channels and adjust the filter parameters while displaying the filters.

The *Display* is the area of the GUI that contains the different forms or frames that provide most of the software functionality. Although this feature has been implemented, Matlab 6.5 does not support multiple frames in the way C++ does. The implementation required to create Matlab's M-files for each frame. These M-files and subroutines are controlled by a *Main subroutine* and provide certain functions such as: create, hide, show controls, and execute procedures. The main subroutine receives a request of the user to change menus through the *TreeView* control. Then the subroutine requests these m-files to show and hide controls. Once the controls are visible, they become available to the user.

The *Navigation buttons* provide independent controls to move through menus without using the *TreeView* control. They were designed to allow SCP code to be run in a different operating system. They also provide an *Exit button*.

7.2 SCP Functions/Processes

SCP version 1.0 has several menus shown in the *TreeView* control.

Load menu allows the user to select a *bin* file. The file contains binary information, but provides no description of the data. The information is ordered by real

and imaginary part, by channel and sample number. Table 7.1 shows the order of a complex, 2-source format.

Table 7.1 Bin (Binary) File Format

Sample 1				Sample 2			
Source 1		Source 2		Source		Source	
Real	Imag	Real	Imag	Real	Imag	Real	Imag

A *format* file is needed to provide the sampling rate, number of input sources, the data type and length. The data length can be a signed 8, 16, 32 or 64 bit integer. An example of a format file contains the following Matlab code.

```
% Format for Binary Files
samplingrate = 6400000;          % 6.4 MSamples/second
complexflag = 1;                 % 1 for complex/ 0 for real
precision    = 'int16';          % 'int8', 'int16', 'int32', 'int64'
sourcechannels = 2;              % Number of sources
```

The software allows the user to modify the sampling rate as convenient. All other parameters must be modified through editing the format file.

The *Display menu* allows the user to see the spectrum of the loaded data as it changes in time. The capabilities include:

- play and stop buttons,
- advance in time using a slider control,
- change the frequency and amplitude limits (*X-Axis*, *Y-Axis* Submenu),
- zoom in/out using mouse click,
- increase display resolution (*Display* menu), and
- increase time interval for refreshing data (*Display* menu).

The *Filter Design menu* allows the user to design a polyphase filter. It provides the following capabilities:

- Filter Design base on specifications,

- *Channel finder* tool to estimate the number of FDM channels, and
- *Filter Adjust* tool to rotate the filter in frequency.

The *FDM menu* shows the filter specification and run the polyphase filter subroutine. The menu allows the user to select between a Matlab and C++ implementation.

The *Demodulation menu* is implemented using a *Grid* control. The grid is a Windows® ActiveX component that presents data in a tabular form. Column one has a list of the demultiplexed bin files/channels. The user can select a demodulation type by clicking in the third column. A first click will change from *skip file* to *FM* demodulation. Consecutive clicks will change the demodulation type to *FM Stereo*, *AM*, and *None*. The software will run the subroutines when Process button is pressed. A processed bin file will contain the date and time when the subroutine was executed. The software will open and play the final audio file by making a double-click on the cell that contains the date.

The *Help menu* provides the software documentation on all its features and gives guidance for troubleshooting. A help window will appear when the user clicks in the title of the topic.

7.3 SCP Installation

Several minimum requirements are needed before installing the software.

- ✓ Matlab 6.5 or higher
- ✓ Windows 98 or higher.
- ✓ Audio player for wav-files.
- ✓ TreeView and GridControl ActiveX Components
- ✓ C++ compiler if C++ code implementation is going to be used.

The entire SCP directory that has the Matlab code has to be copied from the disk to the computer. After Matlab is opened the path has to be change to the SCP folder. Then the InstallSCP must be run from Matlab command line.

After a successful installation, the user must run the StartSCP program from Matlab command line. The GUI interface will appear after a short moment. (See Figure 7.1)

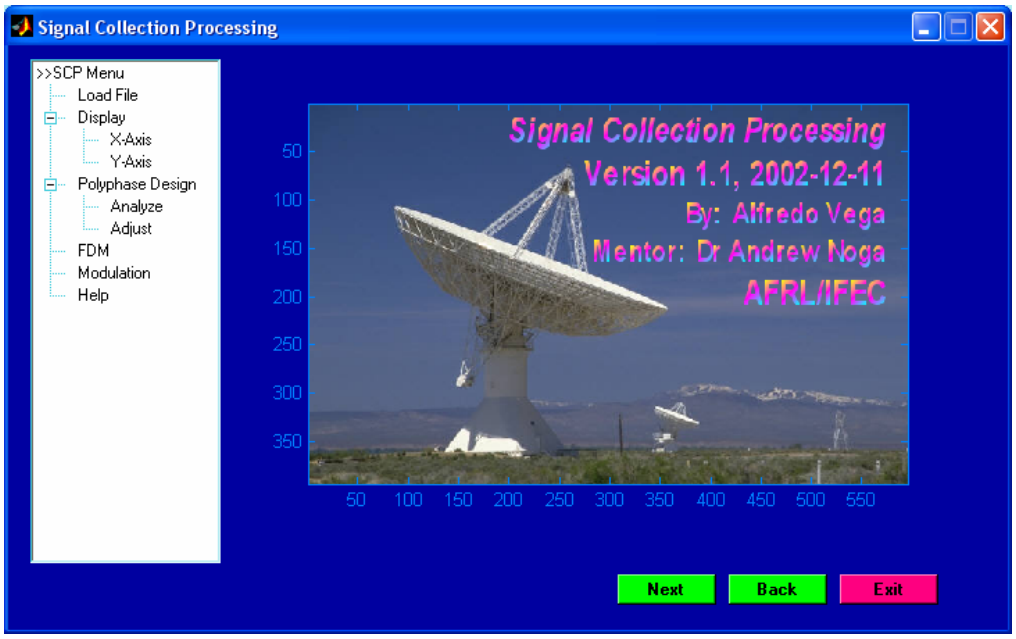


Figure 7.1 SCP Graphical User Interface

7.4 Step by Step FDM using SCP

7.4.1 Loading Data

To change to the Load Menu, press Next button or click Load in the TreeView control. SCP will display a button used for loading files and several fields with properties. (See Figure 7.2) Pressing load will open a dialog window and the user can navigate and find the binary file that contains the data. (See Figure 7.3)

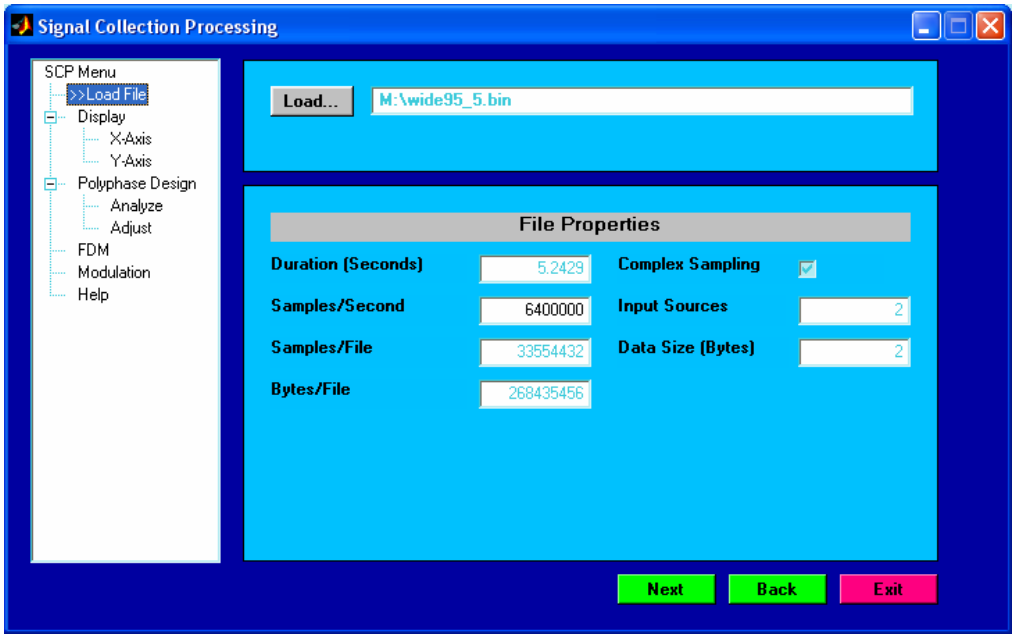


Figure 7.2 Loading Data

After selecting the file a new dialog window will appear asking for a format file. Select the format file with the extension .fmt.txt. The file needs to be created before attempting loading the data. If this step is cancelled, the user will have to attempt to reload the data. Loading the format file will cause the display of important properties such as sampling rate and duration.

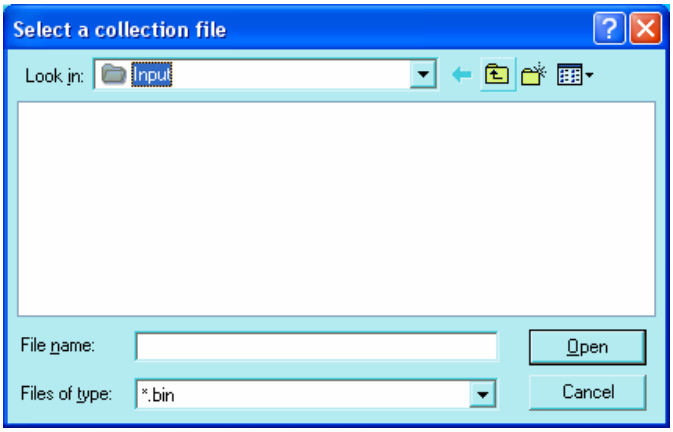


Figure 7.3 Dialog Window for Loading File

7.4.2 Display of Data

The Display menu has several features that control the display of the data.

The Y-Axis submenu contains controls that allow the user to select the input source and display the type of trace. By pressing the Play button, the display shows a time-varying spectrum of the provided signal. Three traces will appear like in Figure 7.4. The blue trace is the current time, the yellow one is the Hold trace and the red one the MaxHold trace.

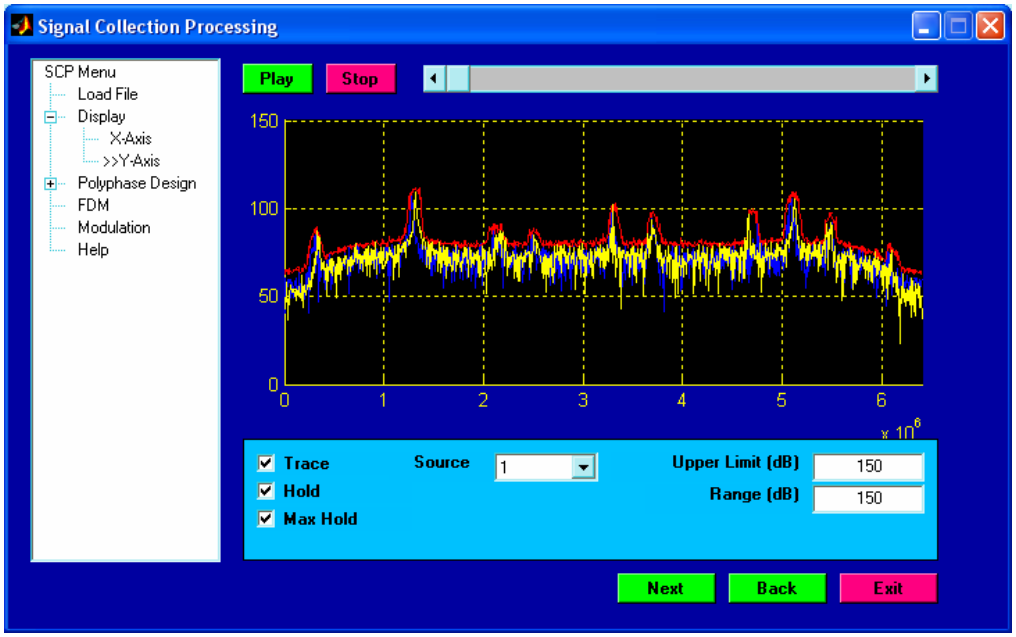


Figure 7.4 Display Menu

While the data is read, the slide bar advances to the right. The process can be paused by pressing stop. Moving the slide backward moves the reading pointer of the file to a previous time.

A graphical visualization of the spectrum helps to determine the position and the number of FDM channels. The next step will be to design a polyphase filter bank.

7.4.3 Polyphase Design

The Polyphase Design menu has a submenu called *Analyze* that provides a tool to find the possible number of channels and their relative frequency offset. By pressing

Channel Finder button, the program will generate a list of possible number of channels and their respective frequency offset.

In the example of Figure 7.5, the program estimated a possible 32 FDM channel filter bank with filters separated every 200 kHz. The rotated the spectrum estimation is 40 percent of the decimated bandwidth.

The program calculated additional design parameters. They are shown in the *Design* menu. (See Figure 7.6) The parameters can be modified manually. By pressing the *Design* button, the program will generate the filter corresponding to the first channel.

Adjust menu provides controls to visualize the filter bank and perform adjustments. Pressing the spinning button up or down will change the channels and frequency offset. (See Figure 7.7)

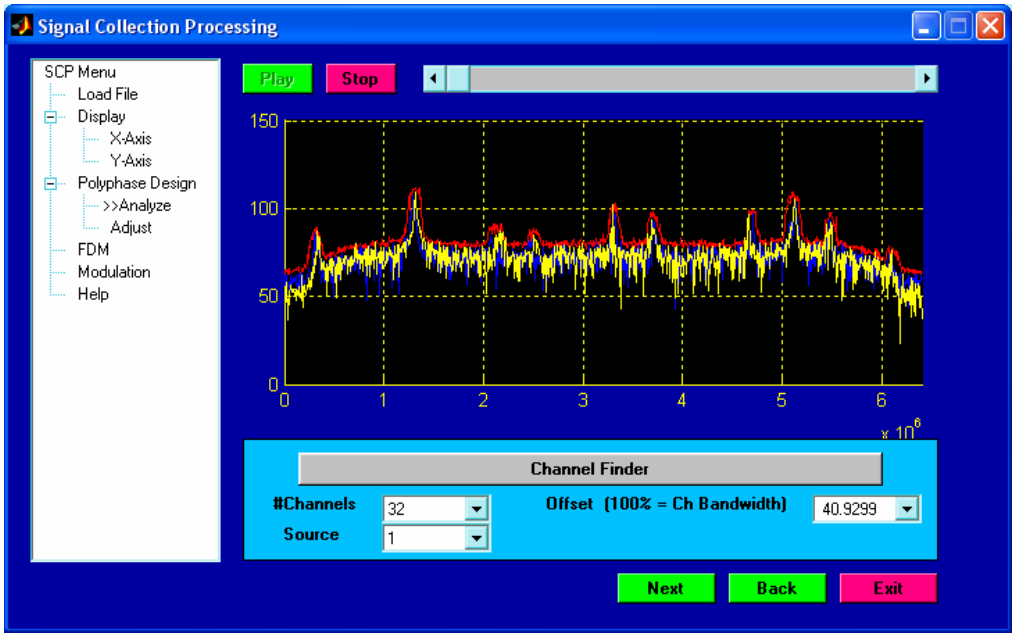


Figure 7.5 Channel Estimation Tool

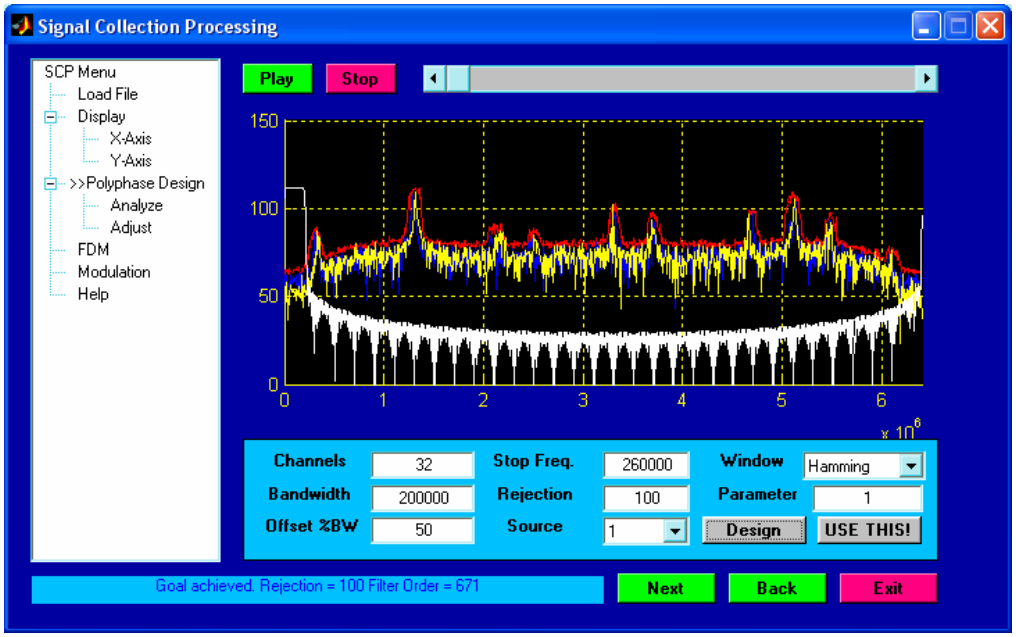


Figure 7.6 Polyphase Filter Design Window

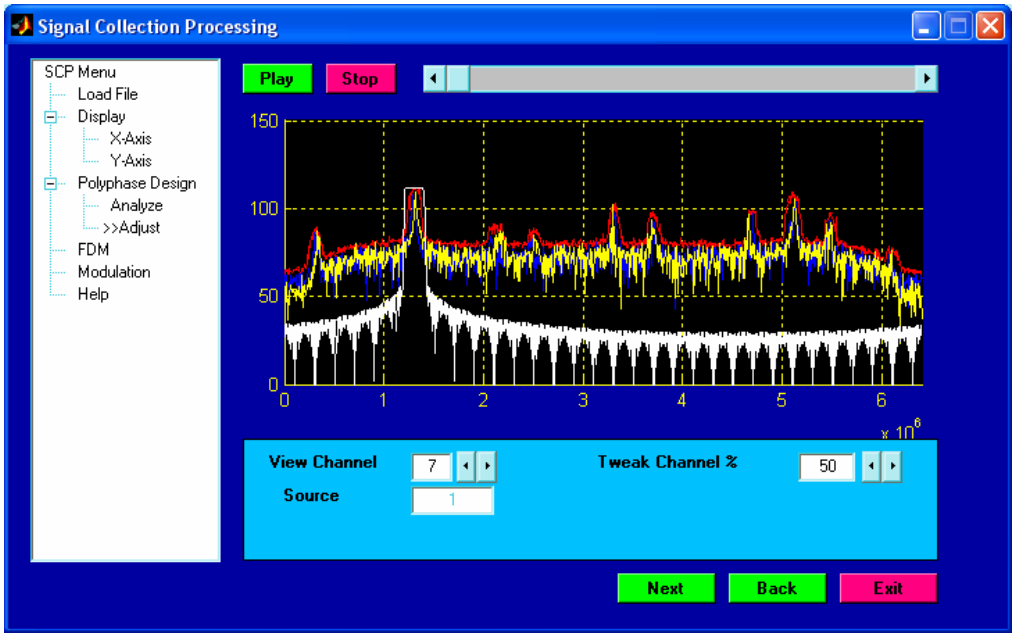


Figure 7.7 Adjust Menu

Once the parameters are fixed, the user is ready to validate the design. In the *Design* menu, there is a *USE THIS* button that will input the polyphase filter parameters into the program. The menu is automatically changed to the *FDM* menu.

7.4.4 FDM Signal Demultiplexing Process

The *FDM* menu only requires the user to start the polyphase filter subroutine. Changing the filter order and selecting a C++ subroutine is optional. These parameters were mainly design for testing purposes. (See Figure 7.8)

The program will display a window with a waitbar showing the percentage of process completed. A 256 MB file having 2 channels of complex data will take approximately 2 minutes to run all the processing in a 2.2 GHz, dual processor computer. Actual figures depend on computer hardware and the operating system.

7.4.5 Modulation Menu

The *Modulation* menu (See Figure 7.9) provides a *Grid* control containing the names of the files processed. By changing the type of modulation from "skip file" to AM, FM, FM Stereo and FM PLL the user tells the program the files that will be processed.

The next step will be to press *Process Now* button. Several waitbars can appear to indicate the percentage of processing that the program has completed. The resulting audio is saved using wave-file format (*.wav).

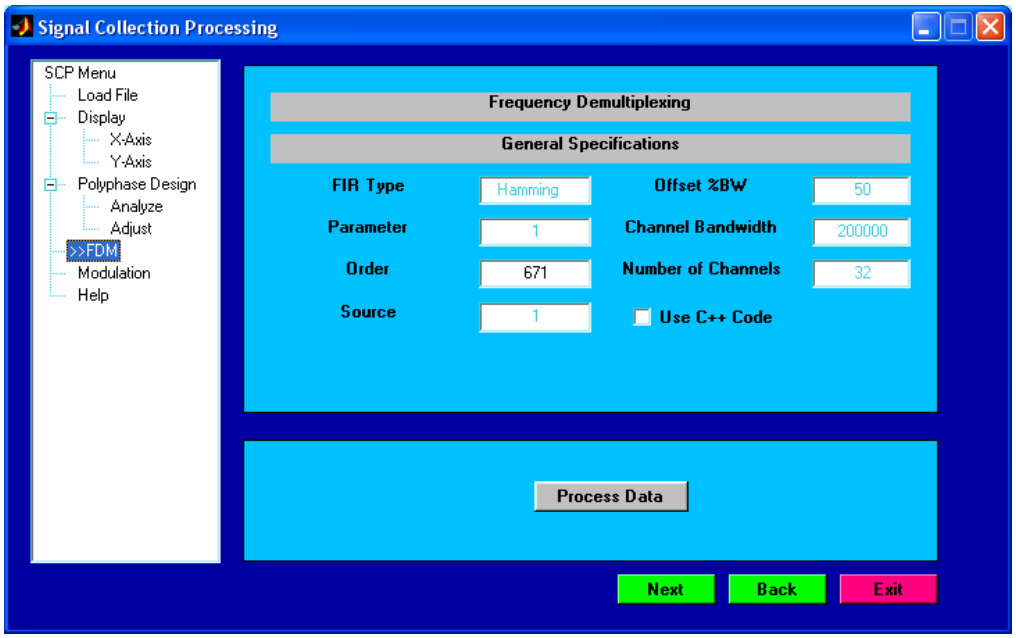


Figure 7.8 FDM Menu

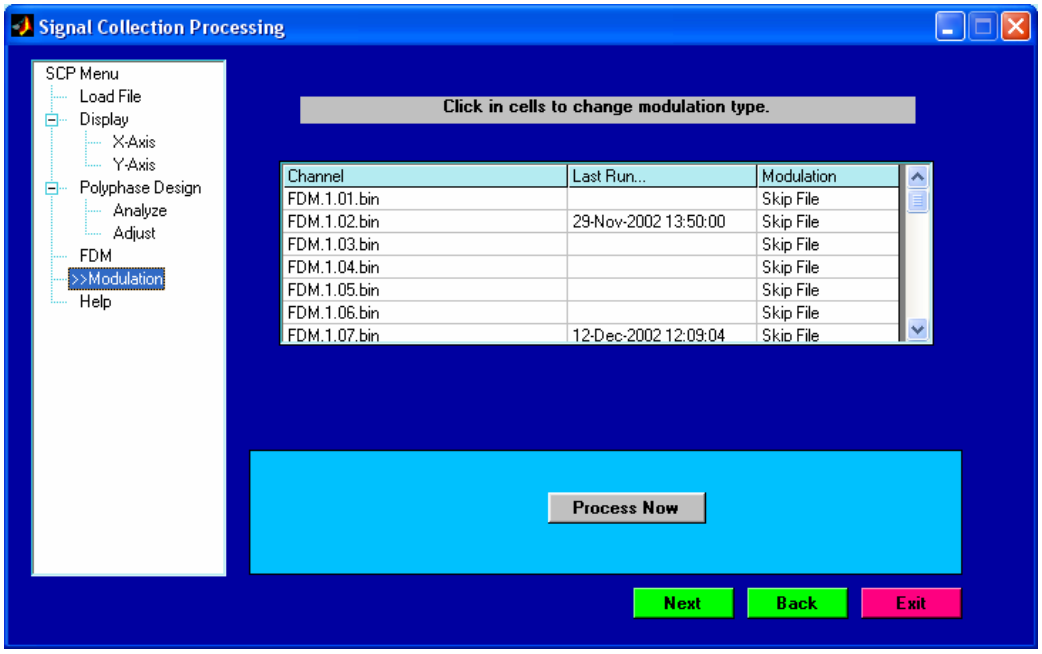


Figure 7.9 Modulation Menu

A double-click in the date that is shown on *Last Run* column will play the audio file corresponding to the selected channel.

8. Summary

The SCP project provides non-real time DSP software that allows users to design and perform fast FDM computations. The project required a research of the Polyphase Filter methods and a comparison to a baseline method. After the concept was validated, a GUI was created to allow the user to customize the design according to the particular specifications. Additional demodulation techniques were explored and have been included in the software to provide the user with supporting tools.

The polyphase filter method performed faster than the baseline approach especially when the number of channels becomes large. Speed improvement was expected due to the usage of DSP methods such as decimation, FFT and complex signal theory used in the polyphase filter algorithm. Also it was noticed that frequency rotation, complex filters, amplitude and phase extraction has contributed to the code simplification.

The polyphase algorithm was coded in Matlab and C++. The test results indicate that Matlab algorithms for FFT and filtering are optimized. Matlab makes use of the FFTW and overlap-saving algorithms which are optimized versions of the FFT and filtering algorithm respectively. This gives Matlab a significant advantage over C++ code, although it should not be concluded that one Matlab has been found superior in performance.

Finally, the FM Stereo synchronization techniques were compared: one using filters and delays and the other using a PLL approach. Although it required more coding, the synchronization using delays had faster performance than the PLL approach. The PLL main disadvantage is that requires non-linear calculations and makes use of memory components (filters) and as a result each calculation has to be sequential as time progresses. The filter approach can take advantage of the FFT overlap saving algorithm and thus perform faster.

9. References

[1] Hayes, Monson H., *Theory and Problems of Digital Signal Processing*, Schuam Outlines Series, Section 5.3, McGraw Hill, New York, 1999, p. 10.

[2] Shenoix, Kishan, *Digital Signal Processing in Telecommunications*, Prentice hall, NJ, 1995, p. 425-427.

[3] Malvar, Henrique S., *Signal Processing with Lapped Transformations*, Artech House Inc., MA, 1992, p. 97-99.

[4] Gumas, Charles C., *"Efficient Polyphase Filters Demultiplex FDM Signals"* Personal Engineering, September 1997.

[5] Mitra, Sanjit K., *Digital Signal Processing*, A Computer based approach, McGraw Hill, NY, 2000 p. 48,698-700.

[6] *www.fftw.org* (December 18, 2002)

[7] Lynn, Paul A., *Introductory Digital Signal Processing with Computer Applications*, J. Wiley & Sons, 1994, p. 270-279.

[8] Haykin, Simon S., *Communication Systems,* 4th Ed. New York: Wiley, 2000. p. 124-126.

[9] Andrew Noga, *Complex Band-Pass Filters For Analytic Signal Generation and their Application*, In-House Technical Memorandum, AFRL-IF-RS-TM-2001-1, Rome, NY, 2001.
[10] Sinclair, Jim, *Radio Signal Finding*, McGraw Hill, NY, 2001, p. 203-207
[11] Best, Roland E., Phase Locked Loops Theory, Design and Applications, McGraw-Hill, New York, 1984, p. 69, 255-259.
[12] Rosedranz, Werner, *Design and Optimization of a Digital FM Receiver Using DPLL Techniques*, International Conference on Acoustics Speech and Signal Processing, IEEE, 1985.
[13] Stephens, Donald R., *Phase Locked Loops for Wireless Communications, Digital Analog and Optical Implementations*, Kluwer Academic, Boston, 2002, p. 209-245, 395-408.
[14] Egan, William F., Phase-Lock Basic, John Wiley & Sons, New York, 1998.
[15] Rorabaugh, C., Britton, *Communications Formulas & Algorithms*, McGraw Hill, NY, p. 93-94.